# An 800 Mpixels/s, ~260 LUTs Implementation of the QOI Lossless Image Compression Algorithm and its Improvement through Hilbert Scanning

Vincenzo Liguori Ocean Logic Pty Ltd enzo@ocean-logic.com

**Abstract**

**This paper discusses the hardware implementation of an encoder and a decoder for the QOI lossless RGB image format. Some results are given for the AMD/Xilinx architecture, reaching over 800 Mpixels/s in Ultrascale+ and more than 4K@30 in Artix-7. A compression improvement of up to ~15% is also observed when natural images are scanned along a Hilbert curve instead of the normal raster order.**

## The QOI Algorithm

QOI stands for Quite OK Image Format and it is a fast, simple lossless image compression algorithm.

As it can be seen in these benchmarks, the performance of the algorithm is usually somewhere in between two well known image compression libraries (like libpng and stb) but at a much lower computational cost.

QOI uses a combination of some well known image compression techniques and innovative ideas. Compressed data is byte aligned.

Pixels are compressed as:

- runs of identical pixels
- an index to a 64 pixel cache of previously encountered pixels
- a difference to the previous pixel
- a full RGB pixel

The most innovative part of the algorithm is the pixel cache that is accessed with a simple hash function of the pixel itself.

The QOI algorithm is an excellent compromise between compression performance and low algorithmic complexity. The image viewer Irfanview supports the QOI format.

## QOI Encoder

The QOI encoder described here is based on the following characteristics:

- One 24 bit RGB pixel encoded, guaranteed every clock cycle
- Compressed data output through a 32 bit interface
- Fully synchronous design based on the rising edge of the clock

Since each a pixel can be encoded with up to 4 bytes, the 32 bit output interface is necessary in order to ensure the encoding of a pixel every clock cycle in the worst case.

Supporting RGBA (alpha) pixels can require up to 5 bytes to encode a pixel. This would have required a substantially larger output interface (64 bit wide), something I felt to be unjustified at this stage. Besides, QOI makes no attempt to compress a pixel when alpha changes: it simply outputs 5 bytes (RGBA pixel + code).

The design does not output a QOI header, although this would be very simple to add.

The QOI encoder is very simple to operate. Once activated, after a delay of 62 cycles, a full 24 bit RGB pixel is accepted every clock cycle. Compression can be suspended at any time by lowering a synchronous enable signal.

This delay is required to clear the pixel cache, as required by the QOI algorithm. In FPGA the pixel cache is implemented as distributed RAM and the latter cannot be cleared in a single clock cycle. In an ASIC, if the pixel cache is implemented as flip-flops, this is certainly a possibility. Realistically though, one needs to consider the fact that 62 cycles at the start are a negligible delay when compared to millions of pixels in an image. In any case, this could be addressed in an FPGA as well.

Once the encoder is started, it will continue to accept pixels every clock cycle until the user signals the last pixel. This will cause the flushing of all the compressed data in the pipeline and the ceasing of operations.

The table below shows the resource utilisation in the AMD/Xilinx architecture in a couple of cases.

| LUTs | Registers | Technology | Speed Grade | Mpixel/s MHz | Equivalent |
|---|---|---|---|---|---|
| 257 | 465 | Kintex Ultrascale+ | -3 | > 800 | > 8K@24 |
| 257 | 465 | Artix 7 | -3 | > 400 | > 4K@48 |

*Table 1: Encoder resources utilisation/pixel rate for AMD/Xilinx.*

Note that the design does not require DPS48 or RAM blocks. The design was placed and routed in isolation on small part and thus timing might be somewhat optimistic. However, it is still representative of the speed that can be obtained.

Preliminary synthesis results for ASIC indicate a size of ~14 Kgates (NAND2 gates, including the pixel caches as flip-flops), running at ~2 GHz in a modern process. This should allow encoding 8K@60 fps.

## QOI Decoder

The QOI decoder design is complementary to the encoder. When the decoder is activated, compressed data is requested from a 32 bit interface. After a few clock cycles delay, 24 bit pixels are decoded and output, one every clock cycle.

Decoding can be suspended at any time by lowering a synchronous enable signal.

Like the encoder, the decoder does not process the QOI header, although this can be easily added.

The table below show the resources utilisation in the AMD/Xilinx architecture in a couple of cases.

| LUTs | Registers | Technology | Speed Grade | Mpixel/s MHz | Equivalent |
|---|---|---|---|---|---|
| 214 | 154 | Kintex Ultrascale+ | -3 | > 800 | > 8K@24 |
| 205 | 154 | Artix 7 | -3 | > 330 | > 4K@37 |

Table 2: Decoder resources utilisation/ pixel rate for AMD/Xilinx.

Again, note that the design does not require DPS48 or RAM blocks.

## Improvements: a Couple of Low Hanging Fruit

It is easy to verify that QOI files can be further compressed if followed by another lossless stage like GZIP or BZIP2. This works very well and their combination with QOI can often result in smaller files than PNG.

This step is only conceptually simple since algorithms like GZIP or BZIP2 are anything but, especially in hardware.

There is, however, a step simple to implement, even in hardware, that will generally improve compression, albeit not as much as the above. The step consists of pre-processing the input image by scanning its pixels along a Hilbert curve rather then the usual raster order. This has the effect to partially (and reversibly) re-order pixels so that similar ones are now grouped together. This takes advantage of the fact that, in natural images, close pixels tend to be more similar to one another. As QOI takes advantage of small differences of adjacent pixels, this should improve compression.

So the steps would be scanning the input image along a Hilbert curve, followed by the QOI algorithm. These steps would be reversed when decoding. Technically, the QOI specification explicitly mentions raster scan order. However, there is nothing in the QOI algorithm itself that requires this as it is only based on previous pixels.

In order to verify this, a simple test was envisioned with Hilbert curves of second, third and fourth order (4x4, 8x8 and 16x16). Five test images, with a mixture of natural images, text and geometric patterns, were used. An example is in Fig.1.

For each Hilbert pattern, each image was clipped so that its dimensions were a multiple of 4, 8 or 16.

A better way of doing this could be extending the image by replicating pixels but, for a quick test, clipping is enough.

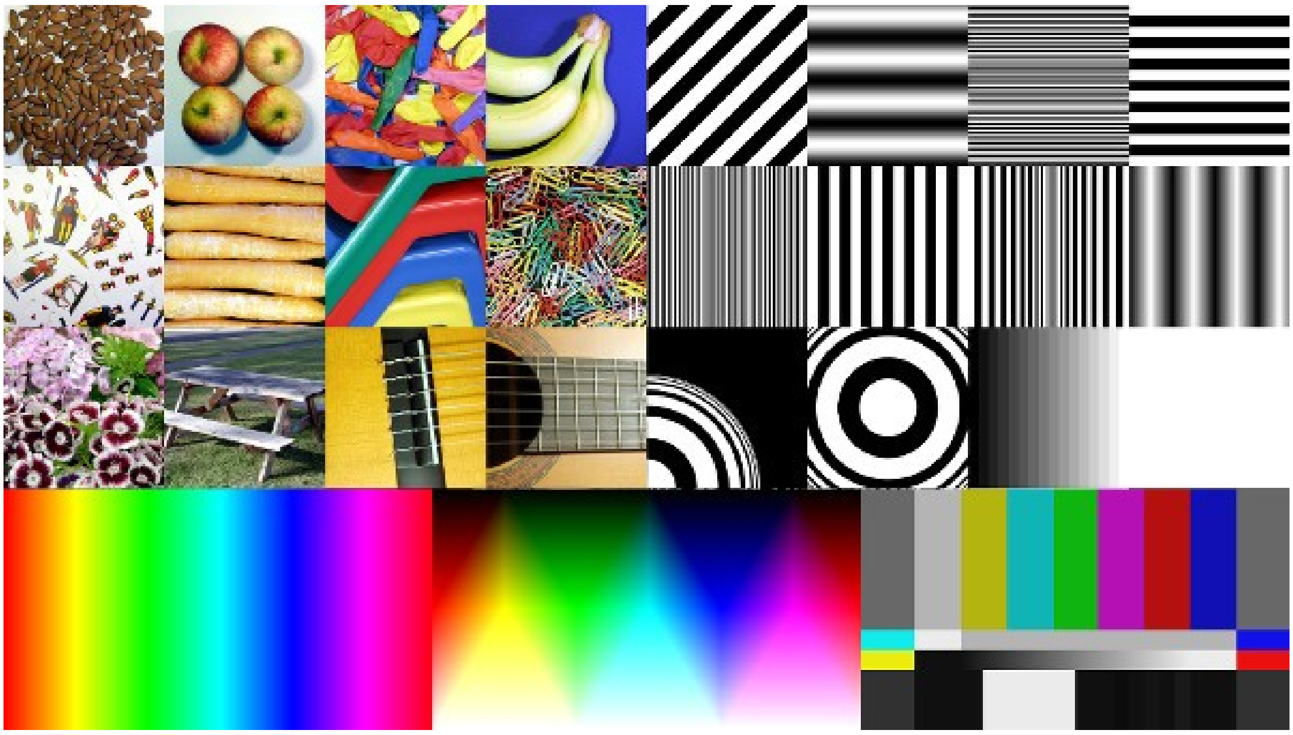
*Figure 1: Test image 2, courtesy of [1].*

After clipping, each image was compressed with QOI. Next, each image was scanned along a Hilbert NxN curve that was placed along a group of N lines in each image (N = 4, 8, 16). Again, this was followed by QOI compression.

| | | | 4x4 | | | 8x8 | | | 16x16 | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| # | Description | Resolution | Q | H+Q | % | Q | H+Q | % | Q | H+Q | % |
| 1 | Similar to Fig 1 + text | 1280x1024 | 1,576,037 | 1,4428,895 | ~10.3 | 1,576,037 | 1,416,594 | ~11.3 | 1,576,037 | 1,383,179 | ~13.9 |
| 2 | See Fig. 1 | 600x338 | 416,288 | 370,787 | ~12.3 | 416,288 | 364,619 | ~14.2 | 415,182 | 357,502 | ~16.1 |
| 3 | Low contrast fish school | 3840x2160 | 8,430,827 | 7,697,965 | ~13.0 | 8,430,827 | 7,329,115 | ~15.0 | 8,430,827 | 7,235,631 | ~16.5 |
| 4 | Two parrots | 768x512 | 675,251 | 604,960 | ~11.6 | 675,251 | 595,606 | ~13.4 | 675,251 | 582,630 | ~16.0 |
| 5 | Sails | 512x768 | 652,383 | 591,701 | ~10.3 | 652,383 | 580,650 | ~12.4 | 652,383 | 568,728 | ~14.7 |

*Table 3: Hilbert scanning test results on natural images.*

The table above shows the results of the test. The image resolution is pre-clipping. For each Hilbert pattern, the compressed image size (in bytes) is indicated (Q), compared to the same clipped image pre-processed with Hilbert scanning (H+Q). The % of improvement is also indicated.

More tests are needed, but it seems that Hilbert scanning improves compression when at least a large portion is a natural or smooth image. However, for artificial images with a few unique colours, there are indications that Hilbert scanning could be detrimental.

From the hardware point of view, Hilbert scanning is a lot simpler than GZIP or BZIP2. All is needed is a double N image lines buffer : while one is written in raster order, the other is read in Hilbert order. Using a convoluted addressing scheme and a dual port RAM it is even possible to use a single N lines buffer.

The downside is that, although simple, a substantial amount of memory might be required, depending on the image size. In FPGA memories are ubiquitous and so, if the design allows it, Hilbert scanning is definitely worth considering.

## Parallelism

Unfortunately, the QOI algorithm is not designed for parallelism. This doesn’t mean that it can’t be done: just divide an image up into tiles, strips, whatever and run multiple encoders simultaneously.

The only problem with this is that compression will be lower as the encoders won’t be able to see each others pixel cash content and will therefore be unable to reference pixels previously seen by other encoders.

Things get more complicated if the various compressed data streams need to be sewed together in a single, legal, QOI file as it would require substantial buffering.

One is faced with similar challenges when parallelising decoding, with the additional complication of knowing the starting point of each tile, strip etc. in the bitstream. In an ordinary QOI file this will be impossible unless such starting points have been noted during encoding. Also, each image portion will also need to be encoded separately in order to avoid the previously outlined pixel cache coherency problems.

In other words, QOI encoding and decoding in parallel is trivial, as long as different image portions are dealt with independently. This can be perfectly acceptable in many applications.

## Applications

The speed and the low resource utilisation of the encoder and decoder allow for many different applications that were previously unpractical.

For example, radiation hardened FPGAs are notoriously slower and less capable than their ordinary counterparts. These designs should allow for much higher pixel rate in aerospace applications.

Another application is compressed frame store to be used, for example, to reduce the bandwidth in case of accessing the reference frames during video compression as in H.264 and H.265.

Although compression will be substantially lower than the silicon proven Ocean Logic compressed frame store H.264 encoder, compression will be lossless and therefore less limiting.

In a compressed frame store, images could be organised as multiple independent tiles that can be independently compressed and decompressed in parallel. Tiles can possibly be better compressed with a Hilbert pattern maybe larger than the one used in the test above.

Other applications can include saving power in small wearable devices whose graphic interface elements (such as icons, backgrounds etc.)  are stored in Non Volatile Memory and need to be decompressed.

## Conclusion

A low footprint hardware implementation of the QOI image format has been presented together with various applications and improvements. Both QOI encoder and decoder are now available as IP cores for licensing.

## References

[1] ASUNI N, GIACHETTI A, "TESTIMAGES: A Large Data Archive For Display and Algorithm Testing", Journal of Graphics Tools, Volume 17, Issue 4, 2015, pages 113-125, DOI:10.1080/2165347X.2015.1024298